Defence Research and Development Canada
Recherche et développement pour la défense Canada

Jitlat: A Jitter and Latency Measurement Tool

Donald McLachlan and André Brind-Amour

The work described in this document was sponsored by the Department of National Defence under Work Unit DRDC ARP 15BW

Defence R&D Canada – Ottawa

Canada

Technical Memorandum
DRDC Ottawa TM 2011-047
May 2011

Jitlat: A Jitter and Latency Measurement Tool

Donald McLachlan
Communications Research Centre

André Brind-Amour
Communications Research Centre

The work described in this document was sponsored by the Department of National Defence under Work Unit DRDC ARP 15BW

Defence R&D Canada – Ottawa

Technical Memorandum
DRDC Ottawa TM 2011-047
May 2011

Principal Author

Original signed by Donald McLachlan

Donald McLachlan

Computer Network Researcher

Approved by

Original signed by Joe Schlesak

Joe Schlesak

Head/Radio Communications Technologies

Approved for release by

Original signed by Chris McMillan

Chris McMillan

DRP Chairman

The work described in this document was sponsored by the Department of National Defence under Work Unit DRDC ARP 15BW.

Abstract

As part of a research effort to demonstrate the feasibility of integrated voice and data on a narrow-band tactical radio network, the voice integrators wanted to know how much latency and jitter to expect over the network path. More importantly, they wanted to know the expected packet loss rate and the packet loss distribution. While it was possible to find tools to measure packet latency, jitter, and loss rates, no tool for measuring packet loss distribution was found.

A tool to measure network performance, as would be experienced by periodic constant bit rate applications, such as voice over IP, was developed. The tool sends a periodic one-way constant bit rate data stream along the network path under test, analyses the received data stream, and displays the results to the user. Along with the usual packet statistics (e.g. number of lost, duplicate, and out-of-sequence packets, min/avg/max latency, min/avg/max jitter, etc.) it displays latency and burstiness (packet latency versus arrival time) plots, and in the case of multiple packet loss events it displays a packet loss-length histogram.

Résumé

Dans le cadre d'un projet de recherche voulant démontrer la faisabilité de l'intégration de voix et données sur un réseau radio tactique à bande passante étroite, les chercheurs responsables de l'intégration de la voix voulaient savoir le degré anticipé de latence et de gigue que pouvait encourir les paquets sur un tel réseau. D'abord et avant tout, ils voulaient savoir quel serait le taux anticipé de perte de paquets ainsi que la distribution de ces pertes de paquets. Alors qu'il était possible de trouver divers outils pour mesurer la latence, la gigue et la le taux de perte de paquets, il n'y avait aucun outil disponible pouvant mesurer la distribution des pertes de paquets.

Un outil a été développé pour mesurer la performance de réseau en simulant le comportement d'applications à transmissions périodiques donnant lieu à un débit binaire constant tel que la voix par IP. Cet outil transmet un flux unidirectionnel de données périodiques à débit binaire constant sur le chemin de réseau sous essai, il analyse le flux de données reçues, et il affiche les résultats à l'utilisateur. En plus de tenir compte des indicateurs statistiques usuels de performance (p. ex. le nombre de paquets perdus, reçus en double, ou reçus hors séquence, la latence min/moy/max, la gigue min/moy/max, etc.), il affiche des graphiques de latence et de sporadicité (latence en fonction du temps d'arrivée) et advenant la perte de plusieurs paquets, il affiche un histogramme représentant la distribution des pertes de paquets en fonction du nombre consécutif de paquets perdus.

This page intentionally left blank.

Executive summary

Jitlat: A Jitter and Latency Measurement Tool

Donald McLachlan, André Brind-Amour; DRDC Ottawa TM 2011-047; Defence R&D Canada – Ottawa; May 2011.

Introduction or background: As part of a research effort to demonstrate the feasibility of integrated voice and data on a narrow-band tactical radio network, the voice integrators wanted to know how much latency and jitter to expect over the network path. More importantly, they wanted to know the expected packet loss rate and the packet loss distribution. While it was possible to find tools to measure packet latency, jitter, and loss rates, no tool for measuring packet loss distribution was found.

Results: A tool to measure network performance, as would be experienced by periodic constant bit rate applications, such as voice over IP, was developed. The tool sends a periodic one-way constant bit rate data stream along the network path under test, analyses the receceived data stream, and displays the results to the user. Along with the usual packet statistics (e.g. number of lost, duplicate, and out-of-sequence packets, min/avg/max latency, min/avg/max jitter, etc.) it displays latency and burstiness (packet latency versus arrival time) plots, and in the event of multiple packet loss it displays a packet loss-length histogram.

Significance: A new tool is now available to measure network effects on the delivery of application data. This tool has already been useful in the development of a TCP Performance Enhancing Proxy, and is being used to help evaluate a new MAC layer for integration of voice and data over narrow-band military radio networks.

Future plans: To release the tool to the open source software community.

Sommaire

Jitlat: A Jitter and Latency Measurement Tool

Donald McLachlan, André Brind-Amour; DRDC Ottawa TM 2011-047; R & D pour la défense Canada – Ottawa; Mai 2011.

Introduction ou contexte: Dans le cadre d'un projet de recherche voulant démontrer la faisabilité de l'intégration de voix et données sur un réseau radio tactique à bande passante étroite, les chercheurs responsables de l'intégration de la voix voulaient savoir le degré anticipé de latence et de gigue que pouvait encourir les paquets sur un tel réseau. D'abord et avant tout, ils voulaient savoir quel serait le taux anticipé de perte de paquets ainsi que la distribution de ces pertes de paquets. Alors qu'il était possible de trouver divers outils pour mesurer la latence, la gigue et la le taux de perte de paquets, il n'y avait aucun outil disponible pouvant mesurer la distribution des pertes de paquets.

Résultats: Un outil a été développé pour mesurer la performance de réseau en simulant le comportement d'applications à transmissions périodiques donnant lieu à un débit binaire constant tel que la voix par IP. Cet outil transmet un flux unidirectionnel de données périodiques à débit binaire constant sur le chemin de réseau sous essai, il analyse le flux de données reçues, et il affiche les résultats à l'utilisateur. En plus de tenir compte des indicateurs statistiques usuels de performance (p. ex. le nombre de paquets perdus, reçus en double, ou reçus hors séquence, la latence min/moy/max, la gigue min/moy/max, etc.), il affiche des graphiques de latence et de sporadicité (latence en fonction du temps d'arrivée) et advenant la perte de plusieurs paquets, il affiche un histogramme représentant la distribution des pertes de paquets en fonction du nombre consécutif de paquets perdus.

Importance: Un nouvel outil est maintenant disponible pour mesurer les effets de réseau sur le temps de remise des données d'application. Cet outil a déjà démontré son utilité dans le développement d'un serveur mandataire améliorant les performances TCP et il est présentement utilisé dans l'évaluation d'une nouvelle couche MAC conçu pour l'intégration de voix et données sur des réseaux radio à bande passante étroite.

Perspectives: Le logiciel sera diffusé à la communauté de logiciel libre.

Table of contents

List of figures

This page intentionally left blank.

1 Introduction

Jitlat is a software tool that uses direct measurement at the application level, with UDP or TCP packets, to help the user evaluate how the network affects the delivery of packets from a constant bit rate (CBR) application. Jitlat focuses on latency[1] and jitter[2] measurements.

Jitlat measures and reports the minimum, average, maximum, standard deviation, median and mode of the latency experienced by packets traversing the path, plots a latency histogram (so the shape of the distribution is visible) and the associated cumulative distribution function.

Jitlat reports the target transmit period, the measured average transmit period, and the measured average receive period (and the corresponding rates in packets per second). It also reports the measured maximum and some percentile jitter values, and plots a burstiness plot (latency versus arrival time).

Like other tools, jitlat detects, measures and reports the number of lost, duplicate and out of sequence packets. Uniquely, when lost, duplicate, or out of sequence packets are detected, jitlat plots a loss length histogram, a duplication length histogram, or an out of sequence distance histogram[3], respectively.

Jitlat saves all the parameters that define the test and all the measurement data collected during the test in a log file. This makes it possible for the user to reanalyse the data at a later time or to perform additional post processing of the data with other programs.

Lastly, jitlat uses least squares linear regression to calculate the slope of the burstiness plot. A non-zero slope or an average receive period significantly greater than the target or average transmit period could, for instance, indicate that there is significant clock skew between the two test hosts or that the test flow data rate exceeds the throughput capacity of the network under test.

This document describes why jitlat was developed, design decisions made along the way, the jitlat protocol, the packet formats, the log file format, and some of the data analysis methods used. Some sample output is presented at the end of this document.

---

[1] Latency, also known as transit time or one-way delay, is the time it takes for a packet to travel from the sender to the receiver.

[2] Jitter, a.k.a. delay variation, is a measure of how much the latency for packets from a single stream from the sender to the receiver varies over time.

[3] More precisely, it plots a re-order density plot as per RFC 5236[11].

2 Background

The Research Network Systems (RNS) group at the Communications Research Centre (CRC) has been developing and testing Data Link Layer and Media Access Control (MAC) protocols for use over narrow band radio links. The availability of test tools to adequately quantify network performance over such non-conventional links has always been an issue.

While working on the HCTCN[4] project, voice coding experts integrated the MELPe vocoder into the Bonephone VOIP application. They asked how much latency and jitter to expect to encounter from the network. More importantly, they wanted to know the packet loss rate and packet loss distribution. In order to properly address packet loss issues, they especially wanted to know if most packet loss events were singleton or burst losses.

While it was possible to find tools to measure latency and jitter and packet loss rates, no tool for measuring packet loss distribution was found. Thus the need for such a tool was identified.

3 Jitlat Design Issues

In any system design, trade-offs are often made. In the design of the jitlat program, there were four such decisions which are worthy of discussion. The first two, the sampling method and the timestamp format, are independent, while the choice of measurement methods and time sources are interrelated.

3.1 Poisson versus Periodic Sampling

Most IETF RFCs related to IP network performance measurements are based on the recommendations of RFC 2330 [1]. RFC 2330 discourages periodic sampling because it can either miss network events which are periodic (unless the sampling period is synchronised with the event period) or periodic test packet transmissions could synchronise with network events exacerbating and exaggerating the effects of those events. Instead RFC 2330 recommends Poisson sampling.

RFC 3432 [2] on the other hand makes a case for periodic sampling for, among other reasons, simulation of constant bit rate (CBR) traffic. RFC 3432 also makes three suggestions to address the concerns outlined in RFC 2330 to mitigate the possible effects of active periodic testing. Specifically, it suggests using random test start times, limiting the test (stream) duration, and limiting the total traffic generated so as not to congest the network under test.

[4] HCTCN (High Capacity Tactical Communications Network) was a DRDC TD project demonstrating the feasibility of integrated voice and data networking over high-speed narrow-band UHF/VHF radio links by using a simple priority based CSMA-CA MAC protocol. Demonstrations included manet ad-hoc networking, GPS position reporting, multicasting, and in-band relaying.

The main design objective of jitlat is to measure the network performance as would be experienced by CBR traffic applications such as the VOIP application used in HCTCN. Thus jitlat is firmly in the camp of RFC 3432 and uses periodic sampling.

Jitlat also follows the recommendations of RFC 3432 to mitigate the possible effects of active periodic testing. Jitlat does not schedule test start times, instead tests are started when initiated by the user, so effectively the test start times are random.[5] Jitlat sends limited test streams that by default are 20 seconds in duration. Whether or not jitlat will congest the network under test largely depends on the capabilities of the network being tested. By default jitlat sends a 43200 bps stream of test traffic.[6] All the test parameters that control the overall bit rate of the test stream can be controlled by the user; so jitlat can be configured to suit the capabilities of the network under test. Jitlat also follows other recommendations from RFC 2330. For example it reports (and records) all the parameters required to reproduce the test, it uses standard SI units (with Engineering prefixes) to describe the tests, all times and timestamps are relative to UTC, etc.

3.2 Timestamp Format

In order to measure latency and jitter, packet transmit and receive times must be recorded. Deciding on the timestamp format was less straightforward than one would think. The goal was to select a format that uses some standard units, provides reasonably high resolution, and is computationally simple and elegant. Ideally the format would either be supported natively on a platform or would be easily converted to/from a native format.

3.2.1 Standard Time Formats

POSIX and Unix-like operating systems support three standard time formats. All three formats consist of a 32-bit count of seconds and a count of sub-seconds since the Epoch (00:00 Universal Coordinated Time, January 1, 1970). In the case of *timeb* structures returned by **ftime()**, the sub-seconds are milliseconds stored in a 16-bit unsigned integer. The *timeval* structure, returned by **gettimeofday()** and **ntp_gettime()**, counts sub-seconds in microseconds and stores them in a signed 32-bit integer. Lastly, the *timespec* structure, returned by **clock_gettime()** and used by **nanosleep()**, counts sub-seconds in nanoseconds and stores them in a 32-bit signed integer[7].

3.2.2 NTP Timestamp Format

Other programs, notably NTP (Network Time Protocol, Version 3) and OWAMP[3] (One-way Active Measurement Protocol), send their timestamps using the NTP timestamp format. As described in [4], "NTP timestamp are represented as a 64-bit unsigned fixed-point number, in seconds relative to 0^h on 1 January 1900. The integer part is the first 32 bits and the fraction part is the last 32 bits. This format allows convenient multiple-precision arithmetic and conversion to Time Protocol representation (seconds), but does complicate the conversion to ICMP Timestamp

---

5 Although jitlat could be started from **cron** for automated data gathering purposes.

6 20 bytes IP header + 8 bytes UDP header + 80 bytes of data per packet, at 50 packets per second; this is the default packet payload length and rate sent by Cisco VOIP phones.

7 Of these routines, Microsoft Windows only supports **ftime**().

message representation (milliseconds). The precision of this representation is about 200 picoseconds, which should be adequate for even the most exotic requirements."[8]

3.2.3 *hrtime_t* Time Format

Sun Microsystems' non standards conforming library routine **gethrtime()** returns time in an *hrtime_t* data type which is a signed 64-bit count of nanoseconds. As such, it can directly represent the highest resolution of the standard *nix time formats listed above, i.e. nanoseconds.

3.2.4 Timestamp Format Selection

If one chooses to use any of the three standard second + sub-seconds time formats, the result of every calculation must be normalised. Another option would be to convert the operands to another format, perform the calculations, and convert the result back to the standard format. All of these manipulations are inelegant and inefficient.

While the NTP Timestamp Format and the NTP Date Format are fixed point and avoid the normalisation issue, using those formats would complicate the conversion to/from any of the standard time formats (e.g. floating point division).

On the other hand, conversion to *hrtime_t* from the standard time formats listed above consists of an integer multiplication of the sub-seconds (times 10^6 for *timeb's* milliseconds and by 10^3 for *timeval's* microseconds) and an integer multiplication and addition (+ seconds * 10^9). From then on, most operations can be done directly on the resulting 64 bit *hrtime_t* values and no further normalisation is required[9].

Since the *hrtime_t* time format clearly met the specified selection criteria, it was selected as the timestamp format for the jitlat application.

3.3 Measurement Methods and Time Sources

This section describes various issues related to latency and jitter measurements that were examined before selecting an appropriate methodology for jitlat.

3.3.1 Latency

Latency is calculated by subtracting the time at which a packet was sent from the sender from the time at which the packet was received by the receiver. In order to obtain accurate values, the clocks of (and thus the timestamps from) the sending and receiving hosts must be synchronised.

---

[8] Apparently 232 picoseconds is not precise enough; NTP Version 4[10] defines a new 128-bit NTP Date Format that is comprised of a 64-bit signed seconds field and a 64-bit fraction field resolving .05 attosecond.

[9] Some exceptions occur, for example, when calculating standard deviation (which requires summing the squares of the *hrtime_t* values, which causes even the 64-bit integers to overflow), and when displaying results on screen.

System clock synchronisation is usually achieved by running NTP on the hosts. NTP occasionally reads the time from GPS time servers and then calls **ntp_adjtime()** to speed up or slow down the host's clock in order to track the reference time obtained from the GPS time servers. Because the host clocks must be adjusted in this manner in order to stay synchronised to the common reference time, any such adjustments themselves induce small but necessary errors in the latency measurements.

3.3.2 Jitter

Jitter is a derived metric that is obtained by subtracting the latency measurement for a reference packet from the latency measurement for the selected packet. Thus any errors induced in the latency measurements by NTP's clock corrections induce errors into the jitter measurements. As long as the relative clock skew is not excessive and the test period is kept short, it may be possible to improve the accuracy of jitter measurements by using timestamps from unsynchronised monotonic clocks.

3.3.2.1 Advantages and Disadvantages of Two Timestamps

Although the use of unadjusted clocks for performing jitter calculations could result in more accurate jitter measurements, its impact on the design of a network performance test tool must also be considered.

The test packets would have to be modified to include a second (unadjusted) timestamp. This would increase the minimum size of the test packet payload, by 8 bytes, to 20 bytes (see section 4.3.4).[10] This in turn could limit the types of protocol traffic that could be emulated by jitlat to protocols with no less than 20 bytes of payload in their packets. With respect to the HCTCN project, the genesis for this test tool, the voice packet payload consisted of an RTP header (12 bytes) plus 4 MELPe frames (28 bytes) for a total payload length of 40 bytes. Thus, for that case, a 20 byte minimum packet payload length (that results in a 48 byte IP packet) would not have been an issue.[11]

Since the adjusted/synchronised (realtime) timestamps from a host may not increment in lock-step with the unadjusted (monotonic) timestamps, it may not be possible to directly correlate the latency results with the jitter results.

3.3.2.2 Jitter Reference Packet

In [5] it is stated that the term jitter means different things in different communities, that the term packet delay variation is more precise and self defining, and therefore the term packet delay

[10] For comparison, from [3], OWAMP test packets have a minimum packet payload of 14 bytes.

[11] Note that IP packets smaller than the minimum Ethernet frame data field length of 46 bytes are padded to that length.

variation is preferred.[12] The document describes several existing packet delay variation measurement methodologies and highlights the two methods that predominate the industry.

Method one, from [6], is called Inter-Packet Delay Variation (IPDV).[13] IPDV defines the reference packet as the previous packet in the sending sequence. This choice of reference packet has some implications. It is possible that for a given packet, that the reference packet may not have arrived yet (due to packet reordering in the network), or may never arrive (due to packet loss in the network). In either case the IPDV value for such a selected packet is undefined. In [7] the problem of selected packets possibly having undefined jitter values is addressed by measuring interarrival jitter. To calculate interarrival jitter, the reference packet is defined as the previous packet in the receive sequence. IPDV and interarrival jitter both produce a double-ended distribution, (i.e. with both positive and negative jitter values) neither extent of which, nor the span of which, directly reflect the packet delay variation or the packet delay distribution of the packet stream.

Method two is called Packet Delay Variation (PDV). PDV defines the reference packet as the packet from the receive stream with the minimum delay (latency). This definition of the reference packet has the following benefits: 1) no received packets have an undefined jitter value, 2) PDV produces a single-ended distribution (all jitter values range between zero and a maximum positive value), 3) the PDV distribution for a flow is identical to the distribution of packet latencies for the flow (with the origin shifted to the value of the minimum packet delay) and so directly reflects the delay variation of the packet stream, 4) the PDV values can be directly used for sizing CBR application de-jitter buffers.

3.3.2.3 Chosen Jitter Measurement Method

The main objective of jitlat is to measure the network performance as experienced by CBR traffic applications. The main use of jitter measurements for CBR applications is for de-jitter buffer sizing. Unlike IPDV measurements, PDV measurements can be used for sizing de-jitter buffers. For these reasons jitlat measures and reports PDV jitter values.

In order to minimise the test packet payload length, jitlat packets contain only synchronised timestamps. The resulting minimum test packet payload is only 12 bytes. This allows jitlat to emulate more types of protocol traffic. Additionally, since the same (realtime) timestamps are used for both latency and jitter measurements, the latency and jitter results can be directly correlated.

---

[12] Never the less, because the word “jitter” is common in the voice coding community (e.g. de-jitter buffer), and because this tool was built to address a need of the voice coders, the word jitter is used in this document.

[13] Also known as “IP Packet Delay Variation”.

3.3.3 Limits of GPS and NTP

In [8], the authors report that "Motorola M12+ GPS receivers can be synchronized to within 10 nanoseconds of each other after calibration."[14] In the same report, of NTP time synchronisation, the authors concluded that "A conservative estimate puts the largest local clock deviation to be around 15 milliseconds. Clearly, NTP accuracy is still lacking comparing to the GPS [...]"

Similarly, in [9], it is claimed that "Using algorithms and atomic time, NTP is able to estimate accurate time to within 10 milliseconds over the Internet and 200 microseconds over local area networks, and can accommodate leap seconds to keep in sync with the Earth's actual rotation. The most current version, NTP.v4, uses 64-bit timestamps, but Version 5 (which is still in development) will begin using 128-bit timestamps, which will astronomically increase timing accuracy."

While the current 64-bit NTP timestamps can resolve 232 picoseconds, the previous articles claim that at best NTP clients can only synchronise to within 200 microseconds. This is 6 orders of magnitude away from what NTP can resolve. Furthermore, when synchronising across the Internet (ie. as round trip time and jitter increase) the ability to accurately estimate time decreases. It is the effects of jitter and latency between the NTP client and server, rather than the precision of the NTP 64-bit timestamps, that limit the NTP client's estimate of the "real" time. So will NTP.v5 with 128-bit timestamps really improve time accuracy that much? Only time will tell.

The accuracy with which the system clocks are synchronised to each other is important when performing network latency measurements. In the case of the radio networks being tested at CRC, the bit rates of the networks are relatively low (less than 1 megabit per second) and the one-way delays being measured are relatively long (> 250 milliseconds). So even if the hosts' system times are in error by as much as 15 milliseconds, the vast majority of the latency reported will be due to the actual network delay rather than to the host clock errors, which in this case corresponds to a 6% error margin. If one were testing on higher speed networks, or networks with lower latencies, e.g. with a measured one-way delay of 25 milliseconds, 15 millisecond of clock error would correspond to a 60% error margin. Thus, it is highly recommended to compare the clock synchronisation error of the NTP clients on the test hosts with the measured latency values, in order to assess the validity of the jitlat results[15].

Time synchronisation is an area of research in itself, and thus specifying a time synchronisation method is beyond the scope of this document. There are many ways one could estimate the clock offset between two NTP servers; three possible options are presented here. One possibility is to use the offset from the command "ntpdate -u -d <timeserver>", but the use of ntpdate is deprecated. Alternatively, one could configure the NTP servers to use each other as timeservers or as peers, and to query the NTP databases with the command "ntpq -n -c peers" to learn the

[14] In discussing "Timestamp Errors", [4] states that "GPS has a potential accuracy in the order of a few nanoseconds; however, various considerations of defense policy may limit accuracy to hundreds of nanoseconds [VAN84]." Although GPS currently has its Selective Availability option (accuracy reducing encryption) disabled, it can be re-enabled at any time by the US Department of Defense.

[15] It is interesting that in [12] it states, "Determining an accurate timestamp "error" is in many ways more difficult than getting a "very good" timestamp."

offset between the two NTP servers. Another option would be to run ntptrace on both systems, and to compare the sum of the 'synch distance' and offset to their respective stratum-1 servers.

4 Jitlat Framework

Jitlat consists of a single binary with two modes of execution, three distinct phases of operation, and two communication connections. The following sections provide more detailed descriptions of these elements.

4.1 Modes of Execution

Jitlat consists of a single executable program. Depending on how the executable is started, it operates either as a client or a server. Some systems refer to these elements as the console and the agent, respectively. The program has been designed such that, for testing purposes, the client and the server can be run on a single host, communicating with each other.

4.2 Phases of Operation

Jitlat has three distinct phases of operation. The first phase is the setup phase during which the test parameters are exchanged between the client and the server. This is followed by the test phase during which the test packets are transmitted. The last phase is the results phase during which the test data is written to a log file and, by default, the data is analysed and the results are displayed to the user.

When the user initiates a test, the jitlat client starts the setup phase by establishing a *control connection* to the server's TCP port 2666. This connection is used to exchange control information (e.g. the test parameters) between the client and the server.

Once the test parameters have been exchanged, the jitlat client and server enter the test phase and establish a *test connection*, and then test packets are transmitted and received via this connection. By default this connection is made to the receiver's UDP port number 2667. The user can use command line options to set the transport protocol to TCP and to set the source and destination port numbers.

After the test phase is completed, the program enters the results phase. If during the test phase the test packets were sent from the client to the server, the *control connection* is used to send the collected test data to the client. The *control connection* is used in this manner (only before the test is started or after the test is completed) to ensure there are no control messages being exchanged while the test is in progress, so as not to influence the network while its performance is being measured.

At this point, the server closes its end of the *control* and *test* connections, and waits for a new *control connection* from a client. In the meantime, the client closes its end of the *control* and *test*

connections, saves the test parameters and data to a log file, and (by default) analyses the test data, and displays the results to the user (as shown in Appendix 1).

4.3 Jitlat Signalling

There are two valid protocol message sequences that are shown below. Control messages are sent via the control connection and are shown in bold. Test packets are sent via the test connection and are shown in regular font. The first valid sequence, shown in Figure 1, occurs when the user requests the server to send a test flow to the client.

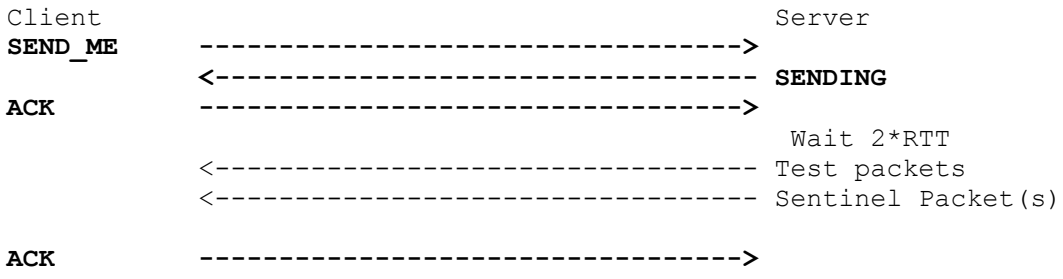

Figure 1. **SEND_ME** flow diagram

The second valid sequence, shown in Figure 2, occurs when the user requests the client to send a test flow to the server.

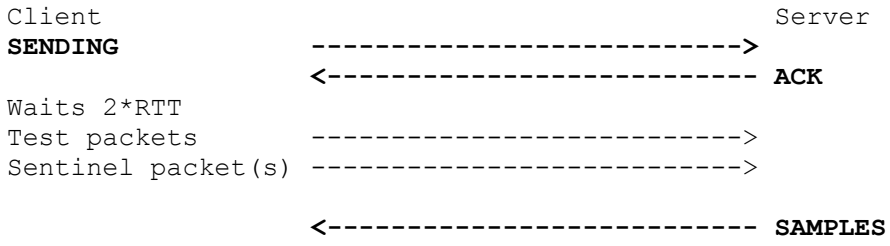

Figure 2. **SENDING** flow diagram

4.3.1 Control Connection Protocol

During the setup phase, a TCP *control connection* is established between the jitlat client and the jitlat server. As shown in Figures 1 and 2, the client and the server exchange a series of control messages via this control connection. Control messages are used not only to determine the

direction of the test flow, but to exchange test parameters and, in the case of the SENDING command sequence, to return the test data to the client.

4.3.2 Control Message Format

The jitlat *control* protocol has a single message type called a *workorder*. *Workorders* are exchanged between the jitlat client and server processes via the control connections, with the data elements in the workorder fields being updated as new information becomes available.[16] The *workorder* message format is shown in Figure 3. Workorder elements are sent in network byte order[17].

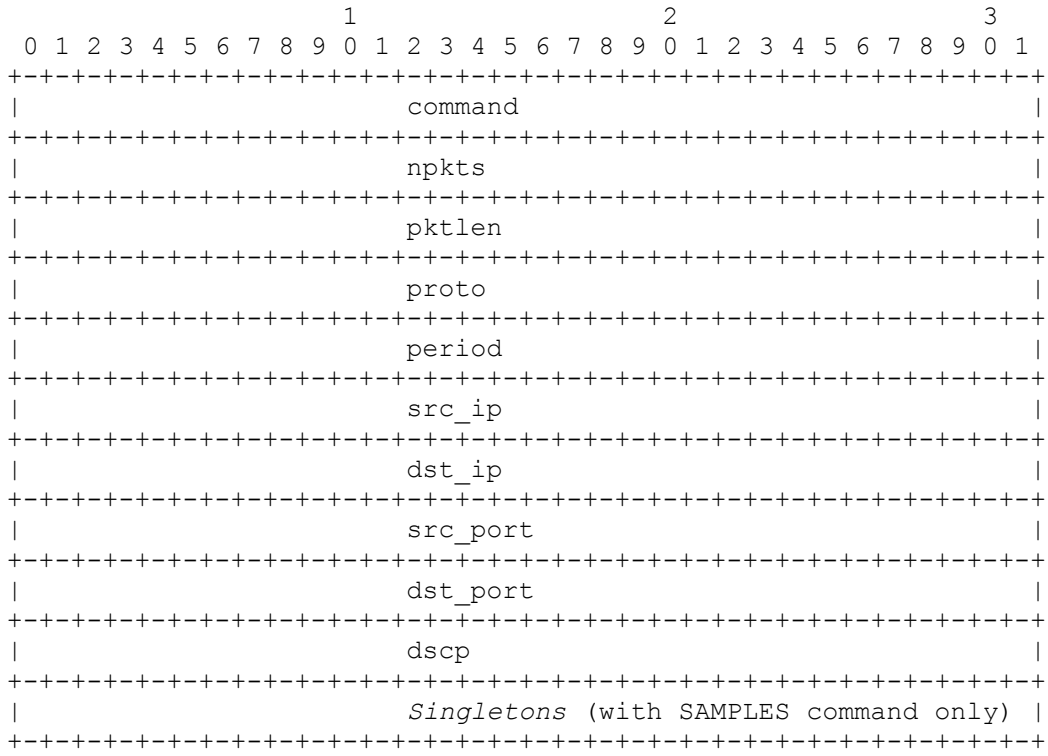

Figure 3. Workorder Message Format

There are six valid commands for the workorder command field. They are SEND_ME, SENDING, SENT, SAMPLES, ACK, and NAK and have the values 1 through 6 respectively. The use of four of these commands is shown in Figures 1 and 2. The fifth value, the NAK (negative acknowledgment) command is used whenever an error condition is detected by either the client or the server. After sending or receiving a NAK, the client and server close the current

[16] e.g. If the user specifies the -R command line option without specifying the src_port, jitlat sends a SEND_ME command to the server with a null src_port value. The jitlat server will update the src_port field with the ephemeral port number allocated by the Operating System, and respond with a SENDING command.

[17] Network byte order is big-endian (i.e. starting with bit number zero, the most significant bit of the most significant byte of the data elements.)

control and test connections. When a test is aborted in this manner, no log file is created and no data analysis is performed. The sixth command, SENT, is currently unused.

The npkts field in the workorder specifies the number of test packets to send via the test connection. By default jitlat sends 1000 test packets. The number of test packets to send can be changed with the -n command line option.

The pktlen field in the workorder specifies the length of the test packet UDP (or TCP) data field (in octets). The default of 80 can be changed with the -l command line option.

The proto field in the workorder specifies the transport protocol to be used for the test connection. The default value of 17 specifies the use of UDP. The -t command line option changes this field to 6 and specifies the use of TCP.[18]

The period field in the workorder specifies the test packet transmission period, expressed in nanoseconds. This defaults to 20000000 (20 ms). This value can be changed with either of the -p (period in nanoseconds) or the -r (rate in packets per second) command line options.[19]

The src_ip and dst_ip fields in the workorder record the IP addresses of the interfaces used to send and receive the test packets. By default the direction of travel is from the client to the server, but this can be reversed with the -R command line option.

By default, the initial value for the src_port field in the workorder is zero. This instructs the sender to update this field with the value of the ephemeral port number assigned to the test connection by the operating system. The user can change the initial value for the src_port field with the -s command line option, instructing the sender to send the test packets from the specified port.

The dst_port field in the workorder specifies the port number to which the sender should send the test packets and from which the receiver should receive the test packets. The default value for this field is 2667, but can be changed with the -d command line option.

The dscp field is the last normal field of the the workorder message. This field specifies the value the sender should put into the test packets' IP headers' DSCP field. The default value is zero, but can be overridden with the -D command line option.

If the command field of the workorder contains the value 4 (the SAMPLES command), it implies Singleton data structures have been appended to the workorder message. The jitlat client reads these Singletons one at a time and saves them in a log file. The format and use of the Singleton data structures and the log files are described in Sections 4.4 and 4.5.

---

[18] When using TCP, because of the larger TCP header, the data rate of the default flow becomes 48000 bps. Also note that the network must be able to support the added traffic load of the TCP ACKs travelling in the reverse direction.

[19] While these options allow the user to specify a transmit period with nanosecond resolution, the host operating system may not be able to schedule events with such accuracy.

4.3.3 Test Connection Protocol

Once the setup phase has completed exchanging the test parameters, i.e. once the first ACK has been received, the test phase commences.

Jitlat uses a separate connection to transmit the test packets. These packets are used to measure the network performance. The sender periodically (using the *period* specified above) builds a test packet, which includes the current realtime and sequence number, and sends the packet to the receiver. The sender continues until *npkts* test packets have been sent. The sender then waits ten test packet periods, and sends Sentinel packets periodically (at one tenth the rate it sent the test packets), until it receives an ACK command from the receiver on the *control connection*.

The receiver waits until it receives a test packet. It then reads the current system realtime and saves these values along with the realtime and the sequence number extracted from the just received packet into a Singleton data structure. The receiver continues processing test packets in this manner until it receives a Sentinel packet[20] from the sender, indicating the end of the test. The receiver acknowledges the end of the test by transmitting either an ACK or a SAMPLES command to the sender via the *control connection.*

4.3.4 Test Packet Format

All the data elements in the test packet (except Padding) are sent in twos-complement format and are sent in network byte order. The test packet payload is 12 bytes long and contains three elements: the Transmit Realtime Timestamp, the Transmit Sequence Number, and (optionally) Padding. The format of the test packets is shown in Figure 4.

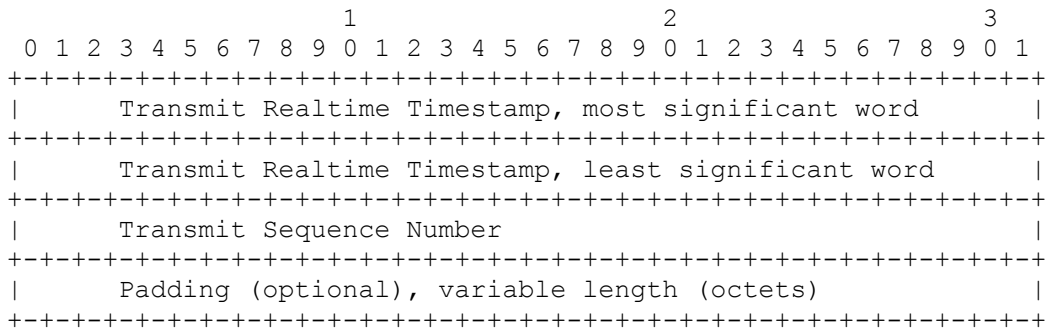

Figure 4. Test Packet Format

Transmit Realtime Timestamp: 64 bits. The Transmit Realtime Timestamp is the sending system's notion of the current time, expressed in nanoseconds since 00:00 Universal Coordinated Time, January 1, 1970. The Transmit Realtime Timestamp is used to measure the latency experienced by the packet. In order to obtain accurate latency measurements, the system time of the two systems at either end of the path under test must be synchronised. This synchronisation is generally achieved by running NTP on the two systems.

[20] In Computer Science a Sentinel is a value used to indicate the beginning or end of a particular block of information. Thus a Sentinel packet is one which contains a Sentinel value.

Transmit Sequence Number: 32 bits. The Transmit Sequence Number represents the order in which the packets were transmitted. The Transmit Sequence Number starts from zero and is incremented by 1 for each new packet sent. The Transmit Sequence Number is used to detect lost and duplicate packets, and to detect packets received out of sequence with respect to their transmission order. Additionally, there is a special sequence number, called a Sentinel that has the value of -1, that is used to signal the completion of a test.

Padding: variable length (octets). When required, Padding octets are appended to the test packets to extend the packet payload up to the size of the packets of the data flow being simulated. In order to defeat any optimisation that might affect the transmission time of the test packets (e.g. compression), the padding bytes are obtained from a random data generator (i.e. **random()**).

4.4 *Singleton* Data Structure

Jitlat defines a singleton data structure that consists of all the information about a received test packet. The format of the *singleton* structure is shown in Figure 5.

```
struct singleton
{
      hrtime_t tx_real_time;
      int32_t  tx_seq;
      hrtime_t rx_real_time;
      int32_t  flags;
}
```

Figure 5. Singleton Data Structure

At the receiver, one singleton data structure is populated for each test packet received. The first two elements of the *singleton* data structure record (in host byte order) the value of the Transmit Realtime Timestamp and the Transmit Sequence Number stored in the test packet by the sender. The next element contains the Receive Realtime Timestamp that was recorded as soon as the test packet was received by the receiver. The last element of the *singleton* structure is used to record anomalous conditions detected during processing. The *singleton* structures for all received test packets are saved in a log file, but only *singletons* with no flags set are used in analysis calculations.

4.5 Log Files

Jitlat saves all the parameters used to generate a test flow and all the *singletons* collected from the test flow in an ASCII file on the client host (where the user initiated the test).[21] The log files allow the results of a test to be re-analysed at a later time using the jitlat -f command line option.

The log file names are generated by **mkstemp()** and consist of the basename “jitlat” followed by a 6 character extension, e.g. “jitlat.Kda4Ib”. The log files consist of an 11 line header followed by

[21] So as not to interfere with the transmission or reception of test packets, the log file is only created after the test phase is completed.

1 line for each test packet received. As an example, the first few lines of the log file "jitlat.Kda4Ib" are shown in Figure 6.

The header lines in the log file start with an octothorpe character (#). The first line contains the date and time (in GMT) when the log file was created. Lines two through eleven of the header consist of the test control fields from the workorder message, described in Section 4.3.2, that fully defines the test scenario.

Each of the remaining lines in the log file each contain the information associated with a single received test packet. These lines consist of five fields. Except for the first field, that has been added for readability and contains the receive sequence number (starting at zero), fields 2 through 5 directly correspond to the members of the *singleton* data structure generated for each received test packet, as described in Section 4.4.

```
# Fri Jul  3 19:45:51 GMT 2009
# command SENDING
# npkts 1000
# pktlen 80
# proto 17
# period 20000000
# src_ip 142.92.34.101
# dst_ip 142.92.34.101
# src_port 33101
# dst_port 2667
# dscp 0
0 1246650331413984400 0 1246650331414098200 0
1 1246650331443879400 1 1246650331443978400 0
2 1246650331463856200 2 1246650331464150000 0
```

Figure 6. Sample Log File

5 Possible Future Work

To assess the validity of the test results, Section 3.3.3 advocates comparing the client and server NTP error estimates with the jitlat latency and jitter results. But at present these values are neither reported by jitlat nor recorded in the log file. The workorder structure could be modified with fields for the client and server NTP error estimates, and these values could be recorded in the log file. Additionally, these values, or the relative time error between them, could be included in the jitlat console output.

As stated in Section 1, jitlat uses direct measurement at the application layer. Because TCP guarantees in-order, error free delivery of data to the application, when testing with TCP, jitlat will not observe any lost, out of sequence, duplicate, or invalid packets. Nevertheless, as shown in Examples 2 and 3 in Appendix 1, jitlat can be useful in analysing TCP application network performance. If jitlat were modified to capture test packets at the Data Link Layer it would be

able to observe out-of-sequence, duplicate, or invalid TCP packets. Still, no “lost” TCP packets would be observed since TCP will retransmit those packets until they are received. But packet loss could possibly be inferred since the latency of any retransmitted packets would be several times that of non-retransmitted packets, and retransmitted packets would arrive out of sequence.

One could implement a GUI that would display the results and plot the graphs in a pane rather than relying on an external program, namely gnuplot, to plot the graphs. The GUI could have a control that allows the user to select which of the available plots should be displayed in the graph pane, and to switch back and forth between them.

This page intentionally left blank.

Appendix 1

This appendix contains three examples of the output from jitlat.

5.1 Example 1

This example shows how to invoke the jitlat program and the console output and graphs from a UDP test flow. In this example, the jitlat client and the jitlat server are running on a single host.

```
janus don> jitlat -n 10000 janus
Connecting to janus. ... Connected.
sending 10000 UDP pkts of 80 bytes from 142.92.34.101:34519 to 142.92.34.101:2667
Data logged to jitlat.0faqgu

target transmit period  = 20.00 ms (50.00  pkts/Sec)
average transmit period = 20.00 ms (50.00  pkts/Sec)
average receive period  = 20.00 ms (50.00  pkts/Sec)

10000 packets sent
10000 packets received
    0 lost (0.00%)
    0 duplicates (0.00%)
    0 invalid (0.00%)
10000 unique/valid packets received
    0 out of sequence (0.00%)

min/avg/max latency = 42.00 us / 74.50 us / 2.871 ms
std dev = 65.94 us
median = 68.40 us
mode at 56.15 us

latency histogram limited to 222.0 us (99 pecentile)
binsize = 2.572 us

Jitter (PDV) percentile:
99.0% = 180.0 us
99.3% = 219.6 us
99.5% = 252.8 us
99.7% = 290.8 us
99.9% = 483.2 us
100%  = 2.829 ms

burstiness linear regression, m = -0.000000, r = -0.004203
```

Figure 7. Console output for Example 1

There is an absolute limit to how quickly packets can transit a network and most packets arrive close to this time. There are many events which can delay the arrival of packets. These effects combine to produce a non-Gaussian delay distribution with a long tail. Thus the mean and median delay can be different from each other, and they can both be significantly different from the mode. This phenomenon is visible in the latency histogram plot in Figure 8.

Also, in contrast to Examples 2 and 3, notice that there are no gaps in packet delivery in the burstiness plot of this example (Figure 9).

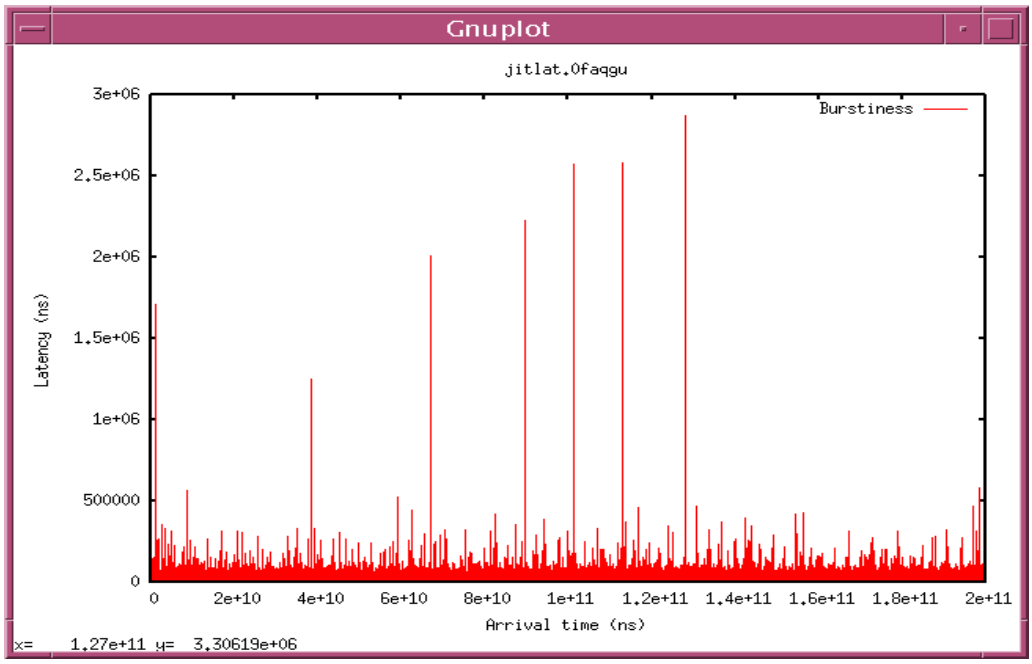

Figure 8. Latency Histogram Plot for Example 1.

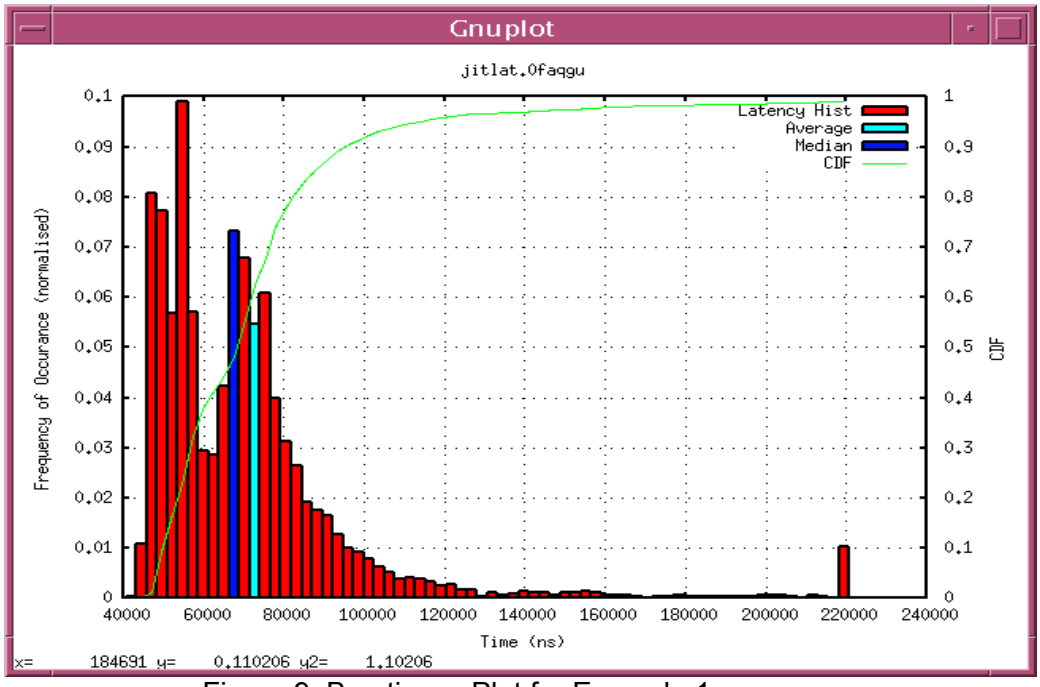

Figure 9. Burstiness Plot for Example 1.

5.2 Example 2

This example shows the console output and graphs produced by re-analysing the contents of a log file from a previous TCP test between two hosts connected via a serial link with a 1.6 second delay.

This example is interesting because it shows the effects of TCP slow start, flow control, and windowing on TCP packet delivery. Notice that although the network latency is 1.6 seconds, the average packet delay is 3.6 seconds and that all the jitter values reported are in excess of 4.8 seconds. Also notice the pauses in the burstiness plot, shown in Figure 12. that occur while the sending TCP stops transmitting and waits for ACKs.

```
janus don> jitlat -f jitlat.KGtbTr
Analysing 1000 TCP pkts of 80 bytes from 192.168.2.4:59844 to 192.168.2.1:2667
# Wed May 27 19:39:11 UTC 2009

target transmit period  = 20.00 ms (50.00  pkts/Sec)
average transmit period = 20.00 ms (49.99  pkts/Sec)
average receive period  = 20.66 ms (48.41  pkts/Sec)

1000 packets sent
1000 packets received
   0 lost (0.00%)
   0 duplicates (0.00%)
   0 invalid (0.00%)
1000 unique/valid packets received
   0 out of sequence (0.00%)

min/avg/max latency = 1.615 s / 3.639 s / 6.692 s
std dev = 1.260 s
median = 3.613 s
mode at 1.628 s

latency histogram limited to 6.512 s (99 pecentile)
binsize = 26.33 ms

Jitter (PDV) percentile:
99.0% = 4.897 s
99.3% = 4.957 s
99.5% = 4.997 s
99.7% = 5.037 s
99.9% = 5.077 s
100%  = 5.077 s

burstiness linear regression, m = -0.133083, r = -0.517128
```

Figure 10. Console output for Example 2

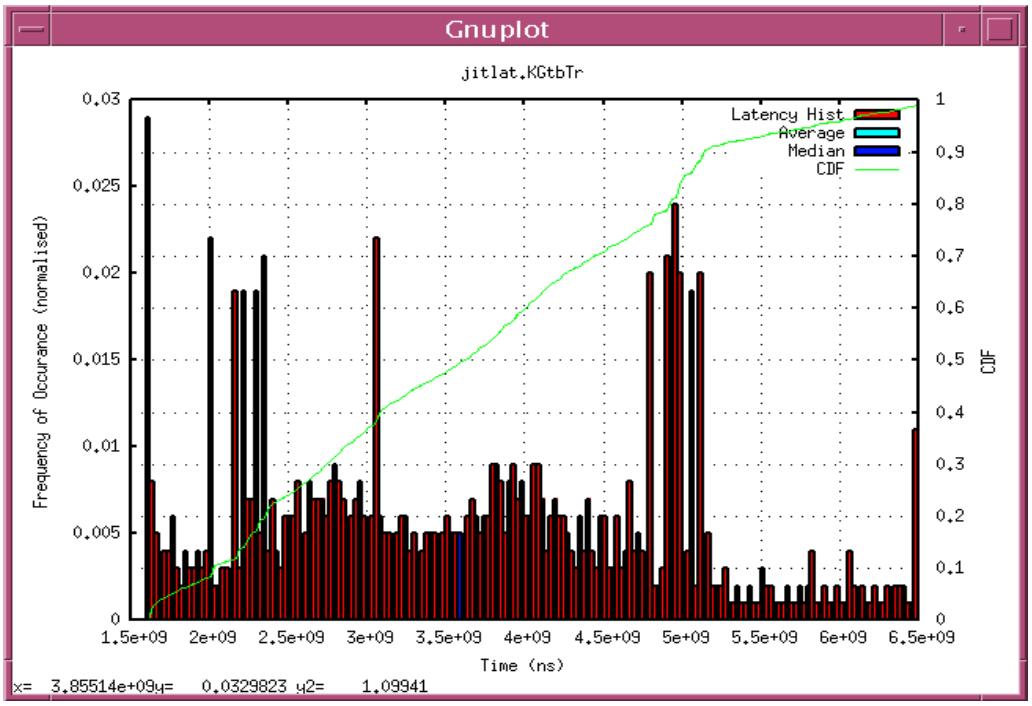

Figure 11. Latency histogram plot for Example 2

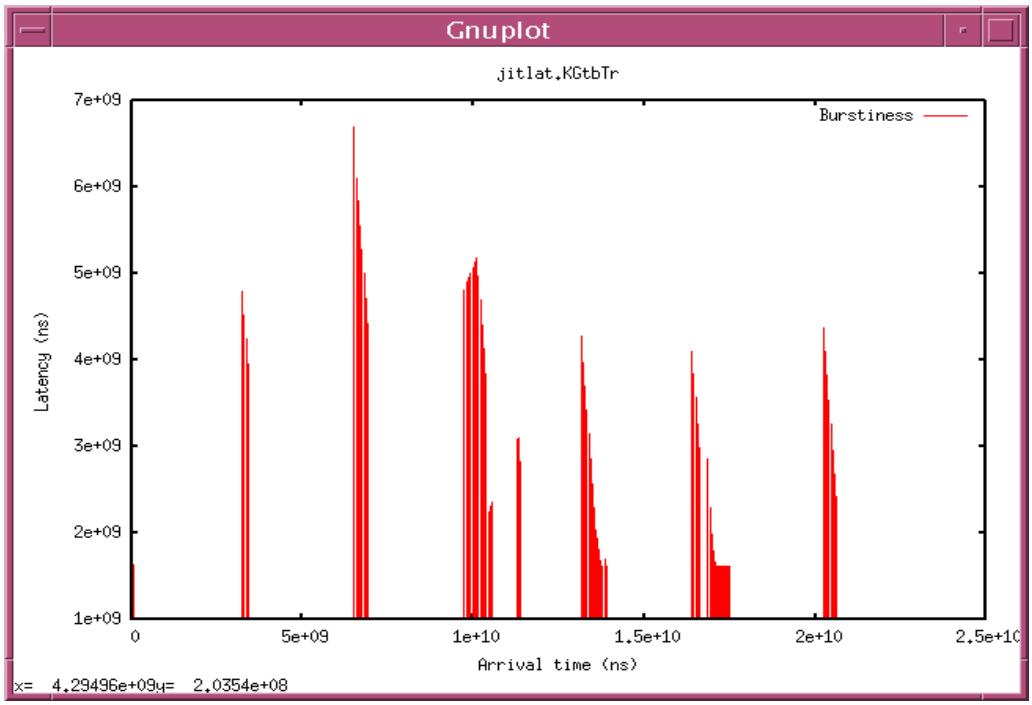

Figure 12. Burstiness plot for Example 2

5.3 Example 3

This example shows the console output and graphs obtained by re-analysing the contents of a log file from a previous test. This test was performed using the same hosts and the same network as in Example 2. In this case, however, a TCP Protocol Enhancing Proxy (PEP) was enabled.

This example demonstrates how the TCP PEP improves the TCP packet delivery. Notice that: the packet delivery in the burstiness plot is much smoother, the average packet latency is now on the order of 1.6 seconds compared to 3.6 seconds with the PEP disabled, the maximum latency is now approximately 1.7 seconds versus 6.7 seconds with the PEP disabled, and the the maximum jitter value is 121.5 milliseconds instead of 5.077 seconds.

The 121 millisecond delay in the burstiness plot was actually an indication that the TCP PEP was not buffering enough data. As a result of this test, the PEP source code was modified to address this issue.

```
janus don> jitlat -f jitlat.ty5cxj
Analysing 1000 TCP pkts of 80 bytes from 192.168.2.4:43875 to 192.168.2.1:2667
# Wed May 27 19:37:00 UTC 2009

target transmit period  = 20.00 ms (50.00  pkts/Sec)
average transmit period = 20.00 ms (49.99  pkts/Sec)
average receive period  = 20.00 ms (50.00  pkts/Sec)

1000 packets sent
1000 packets received
   0 lost (0.00%)
   0 duplicates (0.00%)
   0 invalid (0.00%)
1000 unique/valid packets received
   0 out of sequence (0.00%)

min/avg/max latency = 1.615 s / 1.627 s / 1.737 s
std dev = 26.13 ms
median = 1.615 s
mode at 1.615 s

latency histogram limited to 1.717 s (99 pecentile)
binsize = 204.4 us

Jitter (PDV) percentile:
99.0% = 102.2 ms
99.3% = 102.8 ms
99.5% = 103.1 ms
99.7% = 106.7 ms
99.9% = 121.5 ms
100%  = 121.5 ms

burstiness linear regression, m = -0.000162, r = -0.035737
```

Figure 13. Console output for Example 3

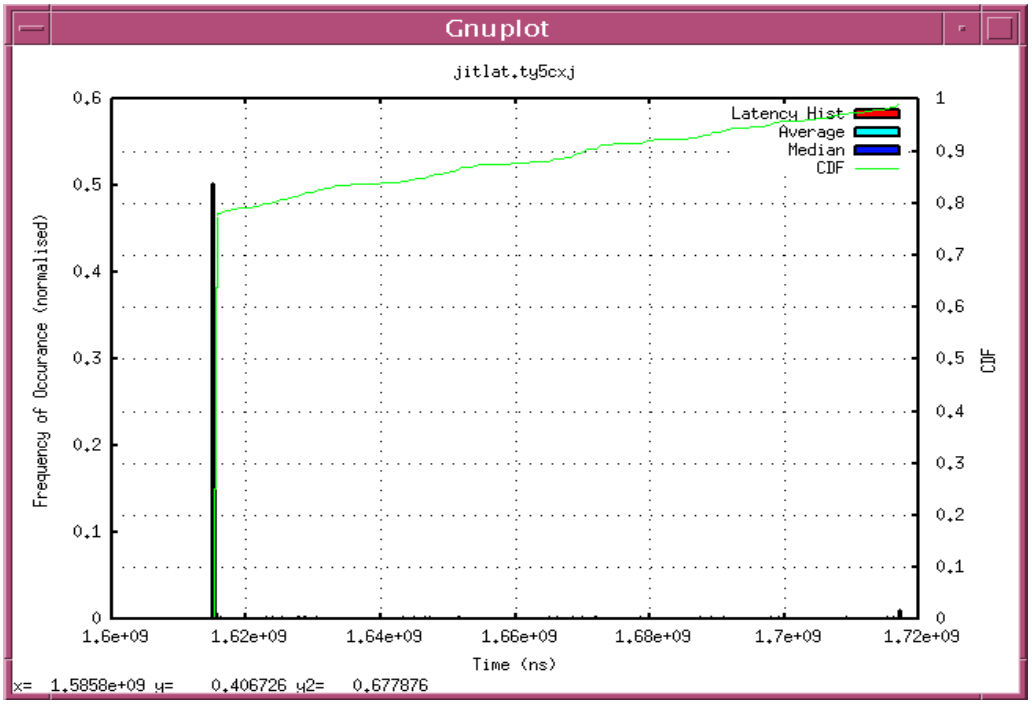

Figure 14. Latency histogram plot for Example 3

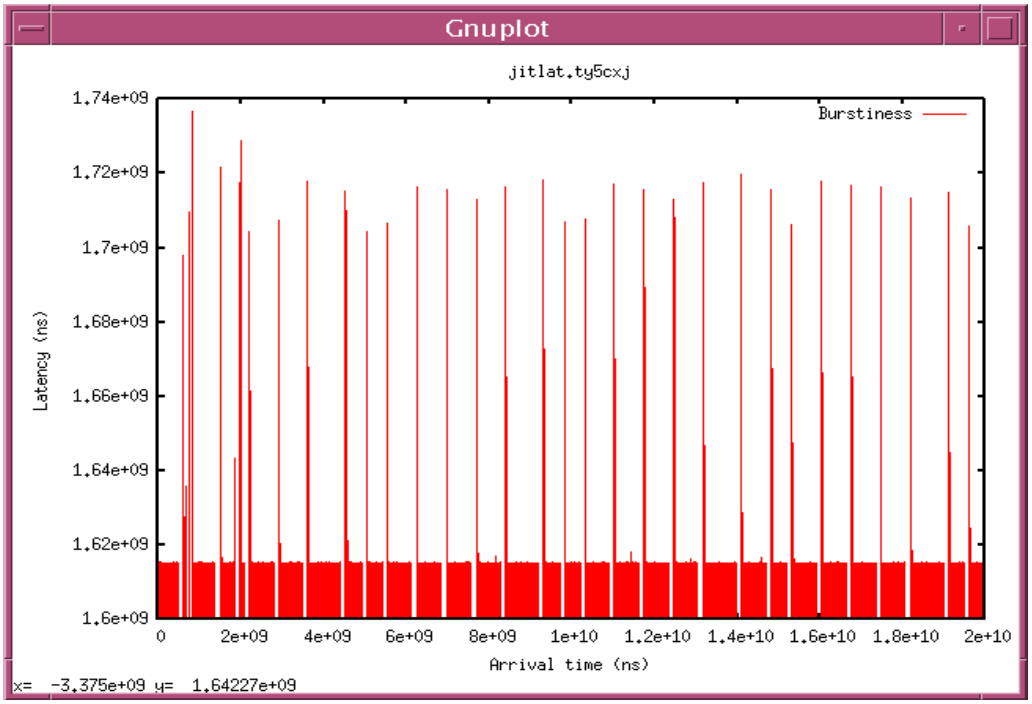

Figure 15. Burstiness plot for Example 3

6 References

[1] Paxson, V., Almes, G., Mahdavi, J., Mathis, M., “Framework for IP Performance Metrics”, RFC 2330, May 1998.

[2] Raisanen, V., Grotefeld, G., Morton, A., “Network performance measurement with periodic streams”, RFC 3432, November 2002.

[3] Shalunov, S., Teitelbaum, B., Karp, A., Boote, J., Zekauskas, M., “A One-way Active Measurement Protocol (OWAMP)”, RFC 4656, September 2006.

[4] Mills, D., “Network Time Protocol (Version 3) Specification, Implementation and Analysis”, RFC 1305, March 1992.

[5] Morton, A., Claise, B., “Packet Delay Variation Applicability Statement”, RFC 5481, March 2009.

[6] Demichelis, C., Chimento, P., “IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)”, RFC 3393, November 2002.

[7] Casner, S., Frederick, R., Jacobson, V., “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, July 2003.

[8] Wang, L., Fernandez, J., Burgett, J., Conners, R., Lui, Y., “An Evaluation of Network Time Protocol for Clock Synchronization in Wide Area Measurements”, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04596234&tag=1

[9] Excel Meridian Data (Storage) Inc, “Network Time Protocol”, http://wiki.emdstorage.com/NetworkProtocols/NetworkTimeProtocol

[10] Mills, D., “Network Time Protocol Version 4 Reference and Implementation Guide”, TR 06-6-1, http://www.ece.udel.edu/~mills/database/reports/ntp4/ntp4.pdf, June 2006.

[11] Jayasumana, A., Pratla, N., Bnka, T., Bare, A., Whitner, R., “Improved Packet Reordering Metrics”, RFC 5236, June 2008.

[12] Boote, J., “Network Performance Tools”, http://www.nanog.org/meetings/nanog43/presentations/Boote_tools_N43.pdf, June 2008.

6.1 Related Documents

Almes, G., Kalidindi, S., Zekauskas, M., “A One-way Delay Metric for IPPM”, RFC 2679, September 1999.

Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., Perser, J., “Packet Reordering Metrics”, RFC 4737, November 2006.

Koodli, R., Ravikanth, R., “One-way Loss Pattern Sample Metrics”, RFC 3357, August 2002.

Almes, G., Kalidindi, S., Zekauskas, M., “A One-way Packet Loss Metric for IPPM”, RFC 2680, September 1999.

Somers, J., Barford, P., Duffield, N., Ron, A., “Improving Accuracy in End-to-end Packet Loss Measurement”, SIGCOMM August 21-26 2005, 1-59593-009-4/05/0008, August 2005.

Anonymous, “Understanding Delay in Packet Voice Networks”, Cisco Document ID 5125, Feb 02, 2006.

Lowekamp, B., Tierney, B., Cottrell, L., Hughew-Jones, R., Kielmann, T., Swany, M., “A Hierarchy of Network Performance Characteristics for Grid Applications and Services”, GWD-R-P (candidate Proposed Recommendation), 24 May 2004.

Stoy, R., Vuagnim, G., Campanella, M., Kudarimoti, L., “Network Performance Metrics MJRA4.1 v1.3”, IST-2002-508833, 22/09/2004.

Mathis, M., Allman, M., “A Framework for Defining Empirical Bulk Transfer Capacity Metrics”, RFC 3148, July 2001.

6.2 Related Software Programs

Boote, J., Karp, A., “One-Way Ping (OWAMP)”, http://www.internet2.edu/performance/owamp/ , 23 January 2009.

Serral, R., “Netmeter”, http://www.cba.upc.edu/netmeter, 2007.

Botta, A., Dainotti, A., Pescapè, A., "Multi-protocol and multi-platform traffic generation and measurement", INFOCOM 2007 DEMO Session, May 2007, Anchorage (Alaska, USA), http://www.grid.unina.it/software/ITG/, May 2007.

Annex A

For users who are familiar with version one of jitalt, this Annexe highlights and explains changes that have been made to jitlat. These changes are based on feedback from early adopters of jitlat, and are most apparent to the user as changes in the displayed results.

A.1 Jitlat Display Changes

Figure 16 shows the new console output. The changes to the console output are:

- The payload size has been removed from the banner line, and is now displayed on the line immediately preceding the target and average periods.
- The target and average period lines now also display the equivalent user throughput rate, based on the payload size, in bits per second.
- The packet statistics display has been rearranged to better show the contribution of the received, duplicate, and invalid packets to the "unique/valid packets" statistic.
- The minimum/average/maximum latency values are displayed down to the full resolution of the timestamp values (i.e. nanoseconds).
- The histogram bin size is no longer displayed.
- NEW: The estimated control channel minimum round trip time is displayed.
- NEW: The estimated relative clock error is displayed.

In addition to the console output changes, jitlat now displays a jitter (PDV) distribution plot instead of a latency distribution plot.

A.1.1 The New Jitlat Error Control Values

The two new console output elements displayed were added for error detection and correction purposes.

The use of the first new element, the estimated control channel round trip time, is straight forward. In general, assuming symmetric path latency, the RTT is usually between two to three times the minimum latency value.[22] If not, the latency values should be viewed with skepticism . For example, if the minimum latency value is negative or is greater than the round trip time, most likely the system clocks being used by the jitlat client and the jitlat server are unsynchronised.

The second new display element, the estimated relative clock error, can be used in two ways. First, if the absolute value of the estimated relative clock error is "small", it is an indication that the reported latency values are likely reasonably accurate. In cases of large relative clock error, the second possible use of the estimated relative clock error, is to correct the latency values, as will be shown shortly.

[22] The minimum control channel RTT is usually slightly smaller than would be reported by ping. On long latency links, it is usually close to twice the minimum latency value. On low latency links, it can be close to three times the minimum latency value.

In Figure 16, for example, the minimum latency value is 1.614 seconds and the minimum round trip time is 3.231 seconds. By itself this is an indication that the latency values are probably reasonably accurate. This conclusion is supported by the fact that the absolute value of the estimated relative clock error is small (80 microseconds). [N.B. for this test the path between the end systems was configured with a 1613 ms delay in both directions, and NTP was being used to synchronise the system time of both end systems to GPS timeservers.]

Conversely, in Figure 17, the minimum latency value is 32.9 seconds and the minimum round trip time is 3.231 seconds. This is the first indication something is amiss. Additionally, the estimated relative clock error is 31.3 seconds. Thus the reported latency values should not be accepted at face value. [N.B. prior to starting this test, NTP was disabled on the jitlat server end system, and the system time was manually advanced by approximately 30 seconds. In all other ways the test setup remained unchanged.]

```
janus don> jitlat -f jitlat.K3Ku7N
Analysing 1000 UDP pkts from 192.168.2.1:32793 to 192.168.2.4:2667
# Tue Dec  8 15:06:22 UTC 2009

payload = 80 bytes per packet.                          User Throughput
target transmit period  = 20.00 ms (50.00  pkts/Sec = 32.00 kbps)
average transmit period = 20.02 ms (49.94  pkts/Sec = 31.96 kbps)
average receive period  = 20.00 ms (49.99  pkts/Sec = 31.99 kbps)

1000 packets sent
1000 unique/valid packets received
     1000 packets received
        0 duplicate packets (0.00%)
        0 invalid packets (0.00%)
   0 lost packets (0.00%)
   0 out of sequence packets (0.00%)

min/avg/max latency = 1.614710000 s / 1.614831767 s / 1.616258000 s
std dev = 87.92 us
median = 1.615 s
mode at 1.615 s

min/avg/max jitter = 0.000 s / 121.8 us / 1.548 ms
std dev = 87.92 us
median = 111.0 us
mode at 78.21 us

jitter histogram limited to 267.0 us (99 pecentile)

Jitter (PDV) percentile:
99.0% = 267.0 us
99.3% = 281.0 us
99.5% = 318.0 us
99.7% = 358.0 us
99.9% = 1.542 ms
100%  = 1.548 ms

Estimated control channel minimum RTT = 3.231 s
Estimated relative clock error = -80.000 us
burstiness plot linear regression, m = -0.000000, r = -0.005674
```

Figure 16. With NTP clock synchronisation

When faced with results like these, the user may use the estimated relative clock error to correct the latency values. This is done by subtracting the estimated relative clock error value from the latency values. Thus the adjusted minimum latency for the test in Figure 17 is:

```
  32.955267000 s
- 31.340420000 s
  --------------
   1.614847000 s
```

Notice that the adjusted minimum latency value is now very close to the minimum latency value from the test results shown in Figure 16.

Tests which produce negative latency values, indicating that a packet took negative time to travel from point A to point B, are an obvious red flag. Negative latency values can be corrected in exactly the same manner, by subtracting the estimated relative clock error from the reported latency values.

```
janus don> jitlat -f jitlat.bD0nyC
Analysing 1000 UDP pkts from 192.168.2.1:32793 to 192.168.2.4:2667
# Tue Dec  8 15:16:40 UTC 2009

payload = 80 bytes per packet.                          User Throughput
target transmit period  = 20.00 ms (50.00  pkts/Sec = 32.00 kbps)
average transmit period = 20.02 ms (49.94  pkts/Sec = 31.96 kbps)
average receive period  = 20.00 ms (49.99  pkts/Sec = 31.99 kbps)

1000 packets sent
1000 unique/valid packets received
     1000 packets received
        0 duplicate packets (0.00%)
        0 invalid packets (0.00%)
   0 lost packets (0.00%)
   0 out of sequence packets (0.00%)

min/avg/max latency = 32.955267000 s / 32.955395850 s / 32.956567000 s
std dev = 76.93 us
median = 32.96 s
mode at 32.96 s

min/avg/max jitter = 0.000 s / 128.8 us / 1.300 ms
std dev = 76.94 us
median = 122.5 us
mode at 90.31 us

jitter histogram limited to 269.0 us (99 pecentile)

Jitter (PDV) percentile:
99.0% = 269.0 us
99.3% = 281.0 us
99.5% = 292.0 us
99.7% = 406.0 us
99.9% = 1.118 ms
100%  = 1.300 ms

Estimated control channel minimum RTT = 3.231 s
Estimated relative clock error = 31.340420000 s
burstiness plot linear regression, m = 0.000000, r = 0.007193
```

Figure 17. Without NTP clock synchronisation

Lastly, note in Figures 16 and 17 that although the latency values are very different, the jitter values are very similar. It is for this reason, rather than displaying a (possibly misleading) latency distribution plot, that jitlat now displays a jitter (PDV) distribution plot.

A.2 Jitlat Protocol Changes

In order to display the path round trip time and the relative clock error estimates, jitlat had to first obtain those values.

Ping (sending 5 packets) could have been used to obtain an estimate of the round trip time, and ntpdate could have been used to obtain a very good estimate of the relative clock error. These ideas were rejected because jitlat would then have been dependent on three external programs: ping, ntpdate, and NTP.[23] Also, this solution would have been sub-optimal on long latency links. For example, on the network used for Figures 16 and 17, ping and ntpdate would each take four seconds plus one RTT to complete (i.e. 14.461 seconds).

Instead of relying on ping, ntpdate and NTP, the jitlat remote control protocol was modified to obtain the desired values.

First a new workorder *command*, called RTT_PROBE, was defined and was assigned the value 7.

The jitlat client code was modified such that, immediately after connecting to the jitlat server process, it sends five RTT_PROBE workorder messages to the jitlat server via the control channel. Like ping and ntpdate packets, these messages are sent at one second intervals. Just prior to sending each of these messages the jitlat client writes its current system time (T1) in the bytes normally used for the 'pktlen' and 'proto' fields.

The jitlat server code was modified such that, immediately after receiving a connection from a client, when it receives each RTT_PROBE workorder message, it writes its current system time (T2) into the 'period' field of the received workorder structure, and sends the modified workorder structure back to the client.

When the client receives each RTT_PROBE message from the server it records its current system time (T3). It then extracts its original transmit time (T1) from the received message and calculates a round trip time estimate (T3 – T1). The jitlat client then extracts the remote system's transmit time (T2) from the received message, calculates a relative clock error estimate (T2 – T1 + RTT/2) and stores this estimate in an array.

Once all five RTT_PROBE messages are received from the server, the client compares the five RTT estimates and finds the minimum RTT. This value is saved and is displayed in the console output when the jitlat test is complete.

Initially the five clock error estimates were summed, and the average was displayed in the console output. But as with the latency values, discussed in Example 1 of Appendix 1, the average clock error estimate was subject to skewing by outliers. Rather than implementing complex methods to remove outliers, jitlat sorts the array of clock error estimates and displays the median clock error estimate (array[2]).

[23] ntpdate will not work unless NTP is running on the “remote” system. Also, as mentioned in Section 3.3.3, the use of ntpdate is deprecated.

While the RTT and clock offset estimate methods could be improved by using techniques from NTP, so far the current methods seem sufficient.

A.3 Jitlat Log File Changes

In order for the RTT and clock offset values to be available during re-analysis, they had to be stored in the log file. Thus the log file format has changed.

The log file header has been extended to include two new lines of text. The first of these lines contains the text of the relative clock error estimate. The second line contains the text of the estimated minimum control channel round trip time.

A.4 The Error Control Values

It could be argued that since jitlat now has estimates of the control channel minimum RTT and the relative clock error that can be used to detect and correct the latency values, why doesn't jitlat do the correction automatically (and plot the latency distribution plot)?

It could do that. But given that the "correction" of the results is straightforward, leaving it to the user is not an undue burden. Additionally, in the author's opinion, a lab test tool should report what it measures so the user has an unadulterated view of the phenomena being observed. Otherwise the user may not be aware of exactly what is happening which, at least to some degree, diminishes the value of the test tool.

As a compromise jitlat will implement two new command line options. The first new command line option "-C" will use the "Estimated relative clock error" to adjust the latency values, and will display the "corrected" values in the console output.

The second new command line option "-L" will use the "Estimated relative clock error" to adjust the latency value associated with each received singleton, and will plot a latency distribution plot of these "corrected" values.

Note that due to the way relative clock error is estimated, when using a high latency network with significantly asymmetric path delays and unsynchronised clocks, the "corrected" values may still not, in fact, be correct, only more correct.

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Communications Research Centre
P.O. Box 11490, Station H,
Ottawa, Ontario K2H 8S2

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Jitlat: A Jitter and Latency Measurement Tool

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

McLachlan, D; Brind-Amour, A.

5. DATE OF PUBLICATION (Month and year of publication of document.)

May 2011

6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)

42

6b. NO. OF REFS (Total cited in document.)

12

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Ottawa
3701 Carling Avenue
Ottawa, Ontario K1A 0Z4

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)

DRDC ARP 15BW

9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Ottawa TM 2011-047

10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

As part of a research effort to demonstrate the feasibility of integrated voice and data on a narrow-band tactical radio network, the voice integrators wanted to know how much latency and jitter to expect over the network path. More importantly, they wanted to know the expected packet loss rate and the packet loss distribution. While it was possible to find tools to measure packet latency, jitter, and loss rates, no tool for measuring packet loss distribution was found.

A tool to measure network performance, as would be experienced by periodic constant bit rate applications, such as voice over IP, was developed. The tool sends a periodic one-way constant bit rate data stream along the network path under test, analyses the receceived data stream, and displays the results to the user. Along with the usual packet statistics (e.g. number of lost, duplicate, and out-of-sequence packets, min/avg/max latency, min/avg/max jitter, etc.) it displays latency and burstiness (packet latency versus arrival time) plots, and in the event of multiple packet loss it displays a packet loss-length histogram.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

latency, jitter, packet loss, narrow-band radios, measurement tools

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

www.drdc-rddc.gc.ca